ARMY RESEARCH LABORATORY

Generating Pseudorandom Numbers From Various Distributions Using C++

by Robert J. Yager

ARL-TN-613 **June 2014**

NOTICES

Disclaimers

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TN-613 **June 2014**

Generating Pseudorandom Numbers From Various Distributions Using C++

by Robert J. Yager
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)* June 2014	**2. REPORT TYPE** Final
3. DATES COVERED (From - To)	October 2012–January 2014
4. TITLE AND SUBTITLE	Generating Pseudorandom Numbers From Various Distributions Using C++
5a. CONTRACT NUMBER	
5b. GRANT NUMBER	
5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)	Robert J. Yager
5d. PROJECT NUMBER	
5e. TASK NUMBER	
5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	U.S. Army Research Laboratory ATTN: RDRL-WML-A Aberdeen Proving Ground, MD 21005-5066
8. PERFORMING ORGANIZATION REPORT NUMBER	ARL-TN-613
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	
10. SPONSOR/MONITOR'S ACRONYM(S)	
11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT	Approved for public release; distribution is unlimited.
13. SUPPLEMENTARY NOTES	

14. ABSTRACT

This report documents a set of functions, written in C++, that can be used to generate pseudorandom numbers that have either uniform or normal distributions and pseudorandom integers that have either uniform or Poisson distributions. An implementation of the Mersenne twister algorithm, developed by Matsumoto and Nishimura, is included. The output from the Mersenne twister is used to generate the various distributions through the use of assorted transformation algorithms.

15. SUBJECT TERMS

pseudorandom, algorithm, Mersenne twister, C++, normal, uniform, Poisson

16. SECURITY CLASSIFICATION OF: a. REPORT	b. ABSTRACT	c. THIS PAGE	17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON / 19b. TELEPHONE NUMBER (*Include area code*)
Unclassified	Unclassified	Unclassified	UU	28	Robert J. Yager 410-278-6689

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

Acknowledgments

I would like to thank Mr. Benjamin Flanders and Dr. Benjamin Breech of the U.S. Army Research Laboratory's Weapons and Materials Research Directorate. Mr. Flanders and Dr. Breech provided technical and editorial recommendations that improved the quality of this report.

INTENTIONALLY LEFT BLANK.

1. Introduction

This report documents a set of functions, written in C++, that can be used to generate pseudorandom numbers that have either uniform or normal distributions and pseudorandom integers that have either uniform or Poisson distributions. An implementation of the Mersenne twister algorithm, developed by Matsumoto and Nishimura (*1*), is included. The output from the Mersenne twister is used to generate the various distributions through the use of assorted transformation algorithms.

The functions presented here are offered as compiler-independent alternatives to functions defined by the new C++11 standard (*2*). Although the new standard defines functions for generating pseudorandom numbers with uniform distributions, normal distributions, Poisson distributions, etc., it does not specify which algorithms will be used to generate those pseudorandom numbers. Thus, compilers that conform to the new standard will all generate pseudorandom numbers from the various distributions, but the actual numbers that are generated may differ from compiler to compiler.

The functions presented in this report have been grouped into the yRandom namespace, which is summarized at the end of this report.

2. Generating Pseudorandom Integers Using the Mersenne Twister 19937 Algorithm – The Rand() Function

The Rand() function uses the Mersenne twister 19937 algorithm to generate uniformly distributed pseudorandom integers in the interval $[0,2^{32})$. According to Matsumoto and Nishimura, the algorithm has a period of $2^{19937}–1$ and passes the Diehard tests for statistical randomness.

The state of the Mersenne twister is stored in an array of 625 32-bit unsigned integers, which is passed as an argument to the Rand() function. The initial state of the Mersenne twister can be set using the Initialize() function (see section 3). The Initialize() function uses a user-supplied 32-bit integer to seed a simple pseudorandom number generator that generates the initial state integers. This effectively allows for the creation of multiple pseudorandom number generators (as many as 2^{32}) that can each produce independent, reproducible sequences of pseudorandom integers.

The code contained in the Rand() function differs from the code that is presented by Matsumoto and Nishimura. To avoid using the modulo operator, their code calculates all of the algorithm's 624 unsigned state integers once out of every 624 function calls. In contrast, the Rand() function uses the ternary operator to avoid using the modulo operator.

2.1 Rand() Code

```
 template<class T>T Rand(//<=====================MERSENNE TWISTER (19937) PRNG
     T I[625]){//<-STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
   T i=I[624],j=i<623?i+1:0,y=I[i]&0x80000000|I[j]&0x7fffffff;
   y=I[i]=I[i<227?i+397:i-227]^y>>1^(y&1)*0x9908b0df,I[624]=j;
   return y^(y^=(y^=(y^=y>>11)<<7&0x9d2c5680)<<15&0xefc60000)>>18;
 }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

2.2 Rand() Template Classes

T **T** should be a 32-bit unsigned integer type. Greater than 32-bit unsigned integer types can be used, but performance may suffer.

2.3 Rand() Parameters

I **I** points to a 625-element array that contains the state of the Mersenne twister algorithm. Each time the Rand() function is called, the array pointed to by **I** is modified. The initial state of the array should be set using the Initialize() function (see section 3).

2.4 Rand() Return Value

The Rand() function returns uniformly distributed pseudorandom integers in the interval $[0,2^{32})$.

2.5 Rand() Example

The following example uses the Rand() function to generate one billion pseudorandom integers. The Initialize() function, introduced in section 3, is used to set the initial state of the Mersenne twister.

```
#include <cstdio>//.....................................................printf()
#include <ctime>//......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//..................................................yRandom
int main(){//<========================EXAMPLE FOR THE yRandom::Rand() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  for(int i=1;i<1000000000;++i)yRandom::Rand(I);
  printf("10^9 pseudorandom numbers generated in %.3f seconds.\nThe 10^9th"
    " number is %u.\n\n",double(clock())/CLOCKS_PER_SEC,yRandom::Rand(I));
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated in 1.778 seconds.
The 10^9th number is 2716480233.
```

2.6 Comparison of the Rand() Function to Other Mersenne Twister Implementations

The following example compares the Rand() function to Matsumoto and Nishimura's genrand() function, as well as to the C++11 built-in Mersenne twister implementation. The example code demonstrates that the three implementations produce identical output (at least for the first 10^9 values and with a seed value of 1). Note that time values will vary based on computer specifications, compiler, compiler settings, etc.

```
#include <cstdio>//.....................................................printf()
#include <ctime>//......................................clock(),CLOCKS_PER_SEC
#include <random>//......................................................MT19937
#include "matsumoto_nishimura.h"//.....................init_genrand(),genrand()
#include "y_random.h"//..................................................yRandom
int main(){//<=====COMPARISON BETWEEN DIFFERENT MERSENNE TWISTER IMPLEMENTATIONS
  double t=0;//.....................................................elapsed time
  init_genrand(1);//..........initialize the Matsumoto-Nishimura implementation
  for(int i=1;i<1000000000;++i)genrand();
  printf("Using Matsumoto and Nishmimura's genrand():\n");
  printf("  10^9 pseudorandom integers generated in %.3f seconds.\n  The 10^9th"
    " number is %u.\n\n",(clock()-t)/CLOCKS_PER_SEC,genrand()),t=clock();
  std::mt19937 g(1);//.............initialize the C++11 built-in implementation
  for(int i=1;i<1000000000;++i)g();
  printf("\nUsing std::mt19937:\n");
  printf("  10^9 pseudorandom integers generated in %.3f seconds.\n  The 10^9th"
    " number is %u.\n\n",(clock()-t)/CLOCKS_PER_SEC,g()),t=clock();
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);// init. yRandom implementation
  for(int i=1;i<1000000000;++i)yRandom::Rand(I);
  printf("\nUsing yRandom::Rand():\n");
  printf("  10^9 pseudorandom integers generated in %.3f seconds.\n  The 10^9th"
    " number is %u.\n\n",(clock()-t)/CLOCKS_PER_SEC,yRandom::Rand(I));
  yRandom::Initialize(I,1),init_genrand(1),g.seed(1);
  bool check=true;
  for(int i=0;i<1000000000;++i){
    unsigned x=yRandom::Rand(I);
    if(x!=genrand()||x!=g())check=false;}
  printf("\nAre the first 10^9 pseudorandom integers generated by\n  Matsumoto "
    "and Nishimura's genrand(), std::mt19937,\n  and yRandom::Rand() identical"
    "? %s\n\n",check?"YES":"NO");
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~28FEB2014~~~~~~
```

OUTPUT:

```
Using Matsumoto and Nishmimura's genrand():
  10^9 pseudorandom integers generated in 3.400 seconds.
  The 10^9th number is 2716480233.


Using std::mt19937:
  10^9 pseudorandom integers generated in 2.231 seconds.
  The 10^9th number is 2716480233.


Using yRandom::Rand():
  10^9 pseudorandom integers generated in 1.857 seconds.
  The 10^9th number is 2716480233.


Are the first 10^9 pseudorandom integers generated by
  Matsumoto and Nishimura's genrand(), std::mt19937,
  and yRandom::Rand() identical?  YES
```

---

3. Initializing the Mersenne Twister – The Initialize() Function

---

The Initialize() function uses an algorithm presented by Nishimura and Matsumoto (*3*) to initialize the 625-element array that is used to store the state of the Mersenne twister algorithm.

3.1 Initialize() Code

```
  template<class T>void Initialize(//<======INITIALIZE STATE OF MERSENNE TWISTER
      T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
      unsigned long s){//<-----------------------------------------SEED [0,2^32)
    I[0]=s&0xffffffff,I[624]=0;
    for(int i=1;i<624;++i)I[i]=(1812433253*(I[i-1]^I[i-1]>>30)+i)&0xffffffff;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

3.2 Initialize() Template Classes

T — **T** should be a 32-bit unsigned integer type. Greater than 32-bit types can be used, but performance may suffer.

3.3 Initialize() Parameters

I — **I** points to storage for a 625-element array that is used to store the state of the Mersenne twister algorithm.

s — **s** specifies the seed for the algorithm that is used to generate the initial state of the Mersenne twister algorithm. **s** can be any value in the interval $[0,2^{32})$.

4. Generating Uniformly Distributed Pseudorandom Numbers – The RandU() Function

The RandU() function can be used to generate uniformly distributed pseudorandom numbers that conform to the probability density function given by equation 1.

$$f(x) = \begin{cases} \dfrac{1}{b-a}, \text{ for } a < x \le b \\ 0, \quad \text{otherwise} \end{cases} \tag{1}$$

Note that the C++ standards document defines a slightly different probability density function for uniformly distributed pseudorandom numbers ($a \le x < b$ rather than $a < x \le b$). The decision to exclude the lower bound from the distribution rather than the upper bound was based on the belief that it would be more likely to help the user to avoid a divide-by-zero error.

The RandU() function uses the transformation presented in equation 2, where r is a uniformly distributed pseudorandom number in the interval (0,1].

$$x = a + (b-a)r \tag{2}$$

Equation 3 can be used to generate r given q, a uniformly distributed pseudorandom integer in the interval $[0,2^{32})$.

$$r = \frac{q+1}{2^{32}} \tag{3}$$

4.1 RandU() Code

```
template<class T>double RandU(//<====UNIFORMLY DISTRIBUTED PSEUDORANDOM DOUBLE
    T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
    double a,double b){//<------------LOWER & UPPER BOUNDARIES OF DISTRIBUTION
  return a+(b-a)*(Rand(I)+1.)/4294967296;//............for a=0 and b=1, (0,1]
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

4.2 RandU() Template Classes

T **T** should be a 32-bit unsigned integer type. Greater than 32-bit types can be used, but performance may suffer.

4.3 RandU() Parameters

I **I** points to an array that contains the state of the Mersenne twister algorithm.

a **a** specifies *a*, the lower bound for the distribution.

b **b** specifies *b*, the upper bound for the distribution.

4.4 RandU() Return Value

The RandU() function returns uniformly distributed pseudorandom numbers in the interval (**a**,**b**]. Note that, due to limitations inherent in the storage of double precision numbers, the return value is not guaranteed to be distinct from **a** in all cases (such as when |**a**| >> |**a**-**b**|).

4.5 RandU() Simple Example

The following example uses the RandU() function to generate and sum one billion uniformly distributed pseudorandom numbers in the interval (2,6].

```
#include <cstdio>//...................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//..................................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandU() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandU(I,2,6);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 8.255 seconds.
Their average is 3.999977.
```

4.6 RandU() Binning Example

The following example uses the RandU() function to generate and bin one billion uniformly distributed pseudorandom numbers in the interval (2,6].

```
#include <cstdio>//.....................................................printf()
#include "y_random.h"//.................................................yRandom
int main(){//<=================BINNING EXAMPLE FOR THE yRandom::RandU() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  int M=1000000000;//......................number of random numbers to generate
  const int N=11;//..................number of bins (not counting overflow bin)
  double B[N];/*<-*/for(int i=0;i<N;++i)B[i]=4*double(i)/(N-1)+2;//.........bins
  double E[N+1]={0};/*<-*/for(int i=1;i<N;++i)E[i]=M/(N-1.);//...expected counts
  int C[N+1];/*<-*/for(int i=0;i<N+1;++i)C[i]=0;//....................raw counts
  for(int i=0,j=0;i<M;++i,++C[j],j=0)
    for(double x=yRandom::RandU(I,2,6);x>B[j]&&j<N;++j);
  printf("                                 COUNT          COUNT\n            ");
  printf("      BIN       ,    (RAW)    ,   (EXPECTED)  ,   %%DIFF \n      ");
  printf("      ---------------------------------------------------------\n");
  for(int i=0;i<N+1;++i){
    if(i==0)printf("                <= %4.1f     ,",B[0]);
    else if(i==N)printf("                 > %4.1f     ,",B[N-1]);
    else printf("            (%4.1f to %4.1f]  ,",B[i-1],B[i]);
    printf("%11d  ,%13.1f",C[i],E[i]);
    E[i]>.1?printf("  ,%9.4f%%\n",fabs(E[i]-C[i])/E[i]*100):printf("\n");}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
                                 COUNT          COUNT
                  BIN       ,    (RAW)    ,   (EXPECTED)  ,   %DIFF
            ---------------------------------------------------------
                <=  2.0     ,          0  ,          0.0
            ( 2.0 to  2.4]  ,   99999019  ,  100000000.0  ,   0.0010%
            ( 2.4 to  2.8]  ,  100010759  ,  100000000.0  ,   0.0108%
            ( 2.8 to  3.2]  ,   99997646  ,  100000000.0  ,   0.0024%
            ( 3.2 to  3.6]  ,  100006913  ,  100000000.0  ,   0.0069%
            ( 3.6 to  4.0]  ,  100010417  ,  100000000.0  ,   0.0104%
            ( 4.0 to  4.4]  ,   99979386  ,  100000000.0  ,   0.0206%
            ( 4.4 to  4.8]  ,   99993880  ,  100000000.0  ,   0.0061%
            ( 4.8 to  5.2]  ,   99999652  ,  100000000.0  ,   0.0003%
            ( 5.2 to  5.6]  ,  100004658  ,  100000000.0  ,   0.0047%
            ( 5.6 to  6.0]  ,   99997670  ,  100000000.0  ,   0.0023%
                 >  6.0     ,          0  ,          0.0
```

5. Generating Normally Distributed Pseudorandom Numbers – The RandN() Function

The RandN() function can be used to generate normally distributed pseudorandom numbers that conform to the probability density function given by equation 4, where μ is the mean of the distribution and σ is the standard deviation.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4}$$

The RandN() function uses the Box-Muller transform (*4*) to generate normally distributed pseudorandom numbers:

$$x = \mu + \sigma\sqrt{-2\ln(r_a)}\cos(2\pi r_b) \tag{5}$$

where r_a and r_b are uniformly distributed pseudorandom numbers in the interval (0,1]. Equation 3 can be used to generate r_a and r_b given q_a and q_b, two uniformly distributed pseudorandom integers in the interval $[0,2^{32})$.

5.1 RandN() Code

```
template<class T>double RandN(//<=====NORMALLY DISTRIBUTED PSEUDORANDOM DOUBLE
    T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
    double m,double s){//<-----------MEAN & STANDARD DEVIATION OF DISTRIBUTION
  return m+s*sqrt(-2*log((Rand(I)+1.)/4294967296))
    *cos(1.4629180792671596E-9*(Rand(I)+1.));
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

5.2 RandN() Template Classes

T — **T** should be a 32-bit unsigned integer type. Greater than 32-bit types can be used, but performance may suffer.

5.3 RandN() Parameters

I — **I** points to an array that contains the state of the Mersenne twister algorithm.

m — **m** specifies μ, the mean of the distribution.

s — **s** specifies σ, the standard deviation of the distribution.

5.4 RandN() Return Value

The RandN() function returns normally distributed pseudorandom numbers.

5.5 RandN() Simple Example

The following example uses the RandN() function to generate and sum one billion normally distributed pseudorandom numbers with μ = 4 and σ = 0.5.

```
#include <cstdio>//.....................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//...................................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandN() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandN(I,4,.5);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 46.213 seconds.
Their average is 3.999998.
```

5.6 RandN() Binning Example

The following example uses the RandN() function to generate and bin one billion normally distributed pseudorandom numbers with μ = 4 and σ = 0.5. The Erf() function is an implementation of an algorithm presented by Abramowitz and Stegun (*5*).

```
#include <cstdio>//.....................................................printf()
#include "y_random.h"//.................................................yRandom
inline double Erf(//<=============================RETURNS THE ERROR FUNCTION OF X
    double x){//<-----------------------------------------------ANY REAL NUMBER
  double t=1/(1+.3275911*fabs(x));//..see Abramowitz and Stegun, 7.1.26 (p. 299)
  double a[]={.254829592,-.284496736,1.421413741,-1.453152027,1.061405429};
  double s=0;/*<-*/for(int i=0;i<5;++i)s+=a[i]*pow(t,i+1);
  return (x<0?-1:1)*(1-s*exp(-x*x));//......................note erf(-x)=-erf(x)
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
int main(){//<==================BINNING EXAMPLE FOR THE yRandom::RandN() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  int M=1000000000;//......................number of random numbers to generate
  const int N=11;//..................number of bins (not counting overflow bin)
  double B[N];/*<-*/for(int i=0;i<N;++i)B[i]=4*double(i)/(N-1)+2;//.........bins
  double E[N+1]={0};/*<-*/E[0]=E[N]=.5*(1+Erf((B[0]-4)/sqrt(2*.5*.5)))*M;//.exp.
  for(int i=1;i<N;++i)E[i]=(.5*(1+Erf((B[i]-4)/
    sqrt(2*.5*.5)))-.5*(1+Erf((B[i-1]-4)/sqrt(2*.5*.5))))*M;
  int C[N+1];/*<-*/for(int i=0;i<N+1;++i)C[i]=0;//....................raw counts
  for(int i=0,j=0;i<M;++i,++C[j],j=0)
    for(double x=yRandom::RandN(I,4,.5);x>B[j]&&j<N;++j);
  printf("                                 COUNT          COUNT\n            ");
  printf("      BIN       ,    (RAW)    ,   (EXPECTED)  ,   %%DIFF \n      ");
  printf("      ---------------------------------------------------------\n");
  for(int i=0;i<N+1;++i){
    if(i==0)printf("                <= %4.1f     ,",B[0]);
    else if(i==N)printf("                 > %4.1f     ,",B[N-1]);
    else printf("            (%4.1f to %4.1f]  ,",B[i-1],B[i]);
    printf("%11d  ,%13.1f",C[i],E[i]);
    E[i]>.1?printf("  ,%9.4f%%\n",fabs(E[i]-C[i])/E[i]*100):printf("\n");}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
                           COUNT          COUNT
      BIN       ,          (RAW)    ,   (EXPECTED)  ,   %DIFF
---------------------------------------------------------
    <=   2.0    ,        31839  ,       31686.0  ,   0.4827%
( 2.0 to  2.4]  ,       655830  ,      655516.1  ,   0.0479%
( 2.4 to  2.8]  ,      7507620  ,     7510327.1  ,   0.0360%
( 2.8 to  3.2]  ,     46597020  ,    46601762.1  ,   0.0102%
( 3.2 to  3.6]  ,    157066112  ,   157056047.3  ,   0.0064%
( 3.6 to  4.0]  ,    288150018  ,   288144661.9  ,   0.0019%
( 4.0 to  4.4]  ,    288145262  ,   288144660.9  ,   0.0002%
( 4.4 to  4.8]  ,    157043835  ,   157056047.3  ,   0.0078%
( 4.8 to  5.2]  ,     46607503  ,    46601762.1  ,   0.0123%
( 5.2 to  5.6]  ,      7508108  ,     7510327.1  ,   0.0295%
( 5.6 to  6.0]  ,       655295  ,      655516.1  ,   0.0337%
     >  6.0     ,        31558  ,       31686.0  ,   0.4041%
```

6. Generating Uniformly Distributed Pseudorandom Integers – The RandI() Function

The RandI() function can be used to generate uniformly distributed pseudorandom integers that conform to the probability density function given by equation 6.

$$f(k) = \begin{cases} \dfrac{1}{b-a+1}, & \text{for } a \le k \le b \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The RandI() function uses the transformation presented in equation 7, where q is a uniformly distributed pseudorandom integer in the interval $[0,2^{32})$.

$$k = a + \operatorname{trunc}\left(\frac{q}{2^{32}}(b-a+1)\right) \tag{7}$$

6.1 RandI() Code

```
template<class T>long RandI(//<=========UNIFORMLY DISTRIBUTED PSEUDORANDOM INT
    T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
    long a,long b){//<----------------LOWER & UPPER BOUNDARIES OF DISTRIBUTION
  return a+T(Rand(I)/4294967296.*(b-a+1));//............................[a,b]
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

6.2 RandI() Template Classes

T — **T** should be a 32-bit unsigned integer type. Greater than 32-bit types can be used, but performance may suffer.

6.3 RandI() Parameters

I — **I** points to an array that contains the state of the Mersenne twister algorithm.

a — **a** specifies a, the lower bound for the distribution.

b — **b** specifies b, the upper bound for the distribution.

6.4 RandI() Return Value

The RandI() function returns a uniformly distributed pseudorandom integer in the interval [**a**,**b**].

6.5 RandI() Simple Example

The following example uses the RandI() function to generate and sum one billion uniformly distributed pseudorandom integers in the interval [–5,4].

```
#include <cstdio>//.....................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//.................................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandI() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandI(I,-5,4);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 10.140 seconds.
Their average is -0.500057.
```

6.6 RandI() Binning Example

The following example uses the RandI() function to generate and bin one billion uniformly distributed pseudorandom integers in the interval [–5,4].

```
#include <cstdio>//.....................................................printf()
#include "y_random.h"//.................................................yRandom
int main(){//<=================BINNING EXAMPLE FOR THE yRandom::RandI() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  int M=1000000000;//......................number of random numbers to generate
  const int N=11;//..................number of bins (not counting overflow bin)
  double B[N];/*<-*/for(int i=0;i<N;++i)B[i]=i-6;//........................bins
  double E[N+1]={0};/*<-*/for(int i=1;i<N;++i)E[i]=M/(N-1.);//...expected counts
  int C[N+1];/*<-*/for(int i=0;i<N+1;++i)C[i]=0;//....................raw counts
  for(int i=0,j=0;i<M;++i,++C[j],j=0)
    for(double x=yRandom::RandI(I,-5,4);x>B[j]&&j<N;++j);
  printf("                                 COUNT          COUNT\n            ");
  printf("      BIN       ,    (RAW)    ,   (EXPECTED)  ,   %%DIFF \n      ");
  printf("      ---------------------------------------------------------\n");
  for(int i=0;i<N+1;++i){
    if(i==0)printf("                <= %4.1f     ,",B[0]);
    else if(i==N)printf("                 > %4.1f     ,",B[N-1]);
    else printf("            (%4.1f to %4.1f]  ,",B[i-1],B[i]);
    printf("%11d  ,%13.1f",C[i],E[i]);
    E[i]>.1?printf("  ,%9.4f%%\n",fabs(E[i]-C[i])/E[i]*100):printf("\n");}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

BIN	COUNT (RAW)	COUNT (EXPECTED)	%DIFF
<= -6.0	0	0.0	
(-6.0 to -5.0]	99999019	100000000.0	0.0010%
(-5.0 to -4.0]	100010759	100000000.0	0.0108%
(-4.0 to -3.0]	99997646	100000000.0	0.0024%
(-3.0 to -2.0]	100006913	100000000.0	0.0069%
(-2.0 to -1.0]	100010417	100000000.0	0.0104%
(-1.0 to 0.0]	99979386	100000000.0	0.0206%
(0.0 to 1.0]	99993881	100000000.0	0.0061%
(1.0 to 2.0]	99999651	100000000.0	0.0003%
(2.0 to 3.0]	100004658	100000000.0	0.0047%
(3.0 to 4.0]	99997670	100000000.0	0.0023%
> 4.0	0	0.0	

7. Generating Poisson-Distributed Pseudorandom Integers – The RandP() Function

The RandP() function can be used to generate Poisson distributed pseudorandom integers that conform to the probability density function given by equation 8, where μ is the mean of the distribution.

$$f(k) = \frac{\mu^k e^{-\mu}}{k!} \tag{8}$$

The RandP() function uses a transformation described by Knuth (*6*) to generate Poisson-distributed pseudorandom integers by finding the largest k that satisfies equation 9.

$$\prod_{i=0}^{k} r_i > e^{-\mu} \tag{9}$$

where r_i is a uniformly distributed pseudorandom number in the interval (0,1]. Equation 3 can be used to generate each r_i given q_i, a uniformly distributed pseudorandom integer in the interval $[0,2^{32})$.

Note that the for large μ, the RandP() function is slow. This problem can be overcome by making use of the fact that, for large μ, Poisson distributions are approximately normal.

7.1 RandP() Code

```
template<class T>T RandP(//<=====POISSON-DISTRIBUTED PSEUDORANDOM UNSIGNED INT
    T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
    double m){//<----------------------------MEAN (MUST BE GREATER THAN ZERO)
  T k=0;/*<-*/for(double P=1,E=exp(-m);P>E;P*=(Rand(I)+1.)/4294967296)++k;
  return k-1;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

7.2 RandP() Template Classes

T — **T** should be a 32-bit unsigned integer type. Greater than 32-bit types can be used, but performance may suffer.

7.3 RandP() Parameters

I — **I** points to an array that contains the state of the Mersenne twister algorithm.

m — **m** specifies μ, the mean of a Poisson distribution.

7.4 RandP() Return Value

The RandP() function returns a Poisson-distributed pseudorandom unsigned integer.

7.5 RandP() Simple Example

The following example uses the RandP() function to generate and sum one billion Poisson-distributed pseudorandom integers with μ = 1.

```
#include <cstdio>//..................................................printf()
#include <ctime>//......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//..................................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandP() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandP(I,1);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 35.708 seconds.
Their average is 1.000013.
```

7.6 RandP() Binning Example

The following example uses the RandP() function to generate and bin one billion Poisson distributed pseudorandom integers with $\mu = 1$.

```
#include <cstdio>//......................................................printf()
#include "y_random.h"//...................................................yRandom
int main(){//<==================BINNING EXAMPLE FOR THE yRandom::RandP() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  int M=1000000000;//.......................number of random numbers to generate
  const int N=11;//...................number of bins (not counting overflow bin)
  double B[N];/*<-*/for(int i=0;i<N;++i)B[i]=i;//............................bins
  double E[N+1]={0};//............................................expected counts
  double F=1;/*<-*/for(int k=0;k<N+1;++k,F*=k)E[k]=pow(1.,k)*exp(-1.)/F*M;
  int C[N+1];/*<-*/for(int i=0;i<N+1;++i)C[i]=0;//....................raw counts
  for(int i=0,j=0;i<M;++i,++C[j],j=0)
    for(double x=yRandom::RandP(I,1);x>B[j]&&j<N;++j);
  printf("                                 COUNT          COUNT\n            ");
  printf("      BIN       ,    (RAW)    ,   (EXPECTED)  ,   %%DIFF \n      ");
  printf("      ----------------------------------------------------------\n");
  for(int i=0;i<N+1;++i){
    if(i==0)printf("                <= %4.1f     ,",B[0]);
    else if(i==N)printf("                 > %4.1f     ,",B[N-1]);
    else printf("            (%4.1f to %4.1f]  ,",B[i-1],B[i]);
    printf("%11d  ,%13.1f",C[i],E[i]);
    E[i]>.1?printf("  ,%9.4f%%\n",fabs(E[i]-C[i])/E[i]*100):printf("\n");}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
                         COUNT          COUNT
      BIN       ,        (RAW)    ,   (EXPECTED)  ,   %DIFF
----------------------------------------------------------
    <=  0.0     ,    367886090  ,  367879441.2  ,   0.0018%
( 0.0 to  1.0]  ,    367871283  ,  367879441.2  ,   0.0022%
( 1.0 to  2.0]  ,    183927644  ,  183939720.6  ,   0.0066%
( 2.0 to  3.0]  ,     61321837  ,   61313240.2  ,   0.0140%
( 3.0 to  4.0]  ,     15332140  ,   15328310.0  ,   0.0250%
( 4.0 to  5.0]  ,      3067729  ,    3065662.0  ,   0.0674%
( 5.0 to  6.0]  ,       510230  ,     510943.7  ,   0.1397%
( 6.0 to  7.0]  ,        72871  ,      72992.0  ,   0.1657%
( 7.0 to  8.0]  ,         9053  ,       9124.0  ,   0.7781%
( 8.0 to  9.0]  ,          999  ,       1013.8  ,   1.4576%
( 9.0 to 10.0]  ,          112  ,        101.4  ,  10.4779%
     > 10.0     ,           12  ,          9.2  ,  30.2061%
```

8. Code Summary

A summary sheet is provided at the end of this report. It presents the yRandom namespace, which contains the six functions that are described in this report.

yRandom Summary

y_random.h

```
#ifndef Y_RANDOM_GUARD//        See Yager, R.J. "Generating Psuedorandom Numbers
#define Y_RANDOM_GUARD//                     from Various Distributions Using C++"
#include <cmath>//....................................exp(),log(),sin(),sqrt()
namespace yRandom{//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
  template<class T>void Initialize(//<======INITIALIZE STATE OF MERSENNE TWISTER
      T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
      unsigned long s){//<----------------------------------------SEED [0,2^32)
    I[0]=s&0xffffffff,I[624]=0;
    for(int i=1;i<624;++i)I[i]=(1812433253*(I[i-1]^I[i-1]>>30)+i)&0xffffffff;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
  template<class T>T Rand(//<=====================MERSENNE TWISTER (19937) PRNG
      T I[625]){//<-STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
    T i=I[624],j=i<623?i+1:0,y=I[i]&0x80000000|I[j]&0x7fffffff;
    y=I[i]=I[i<227?i+397:i-227]^y>>1^(y&1)*0x9908b0df,I[624]=j;
    return y^(y^=(y^=(y^=y>>11)<<7&0x9d2c5680)<<15&0xefc60000)>>18;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
  template<class T>double RandU(//<====UNIFORMLY DISTRIBUTED PSEUDORANDOM DOUBLE
      T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
      double a,double b){//<------------LOWER & UPPER BOUNDARIES OF DISTRIBUTION
    return a+(b-a)*(Rand(I)+1.)/4294967296;//............for a=0 and b=1, (0,1]
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
  template<class T>double RandN(//<=====NORMALLY DISTRIBUTED PSEUDORANDOM DOUBLE
      T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
      double m,double s){//<-----------MEAN & STANDARD DEVIATION OF DISTRIBUTION
    return m+s*sqrt(-2*log((Rand(I)+1.)/4294967296))
      *cos(1.4629180792671596E-9*(Rand(I)+1.));
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
  template<class T>long RandI(//<=========UNIFORMLY DISTRIBUTED PSEUDORANDOM INT
      T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
      long a,long b){//<----------------LOWER & UPPER BOUNDARIES OF DISTRIBUTION
    return a+T(Rand(I)/4294967296.*(b-a+1));//.............................[a,b]
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
  template<class T>T RandP(//<=====POISSON-DISTRIBUTED PSEUDORANDOM UNSIGNED INT
      T I[625],//<--STATE OF MERSENNE TWISTER (FOR T, USE A 32-BIT UNSIGNED INT)
      double m){//<-----------------------------MEAN (MUST BE GREATER THAN ZERO)
    T k=0;/*<-*/for(double P=1,E=exp(-m);P>E;P*=(Rand(I)+1.)/4294967296)++k;
    return k-1;
  }//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
}//@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
#endif
```

RandN() Binning Example

```
#include <cstdio>//...................................................printf()
#include "y_random.h"//...............................................yRandom
inline double Erf(//<============================RETURNS THE ERROR FUNCTION OF X
    double x){//<---------------------------------------------ANY REAL NUMBER
  double t=1/(1+.3275911*fabs(x));//..see Abramowitz and Stegun, 7.1.26 (p. 299)
  double a[]={.254829592,-.284496736,1.421413741,-1.453152027,1.061405429};
  double s=0;/*<-*/for(int i=0;i<5;++i)s+=a[i]*pow(t,i+1);
  return (x<0?-1:1)*(1-s*exp(-x*x));//.....................note erf(-x)=-erf(x)
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
int main(){//<=================BINNING EXAMPLE FOR THE yRandom::RandN() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  int M=1000000000;//......................number of random numbers to generate
  const int N=11;//..................number of bins (not counting overflow bin)
  double B[N];/*<-*/for(int i=0;i<N;++i)B[i]=4*double(i)/(N-1)+2;//.........bins
  double E[N+1]={0};/*<-*/E[0]=E[N]=.5*(1+Erf((B[0]-4)/sqrt(2*.5*.5)))*M;//.exp.
  for(int i=1;i<N;++i)E[i]=(.5*(1+Erf((B[i]-4)/
    sqrt(2*.5*.5)))-.5*(1+Erf((B[i-1]-4)/sqrt(2*.5*.5))))*M;
  int C[N+1];/*<-*/for(int i=0;i<N+1;++i)C[i]=0;//...................raw counts
  for(int i=0,j=0;i<M;++i,++C[j],j=0)
    for(double x=yRandom::RandN(I,4,.5);x>B[j]&&j<N;++j);
  printf("                                 COUNT          COUNT\n           ");
  printf("      BIN        ,     (RAW)     ,   (EXPECTED)   ,   %%DIFF \n      ");
  printf("      -------------------------------------------------------\n");
  for(int i=0;i<N+1;++i){
    if(i==0)printf("                  <= %4.1f      ,",B[0]);
    else if(i==N)printf("                  > %4.1f      ,",B[N-1]);
    else printf("              (%4.1f to %4.1f]  ,",B[i-1],B[i]);
    printf("%11d  ,%13.1f",C[i],E[i]);
    E[i]>.1?printf("  ,%9.4f%%\n",fabs(E[i]-C[i])/E[i]*100):printf("\n");}
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
                                 COUNT          COUNT
           BIN        ,         (RAW)     ,   (EXPECTED)  ,    %DIFF
      ---------------------------------------------------------------
              <=  2.0      ,       31839  ,      31686.0  ,   0.4827%
          ( 2.0 to  2.4]  ,      655830  ,     655516.1  ,   0.0479%
          ( 2.4 to  2.8]  ,     7507620  ,    7510327.1  ,   0.0360%
          ( 2.8 to  3.2]  ,    46597020  ,   46601762.1  ,   0.0102%
          ( 3.2 to  3.6]  ,   157066112  ,  157056047.3  ,   0.0064%
          ( 3.6 to  4.0]  ,   288150018  ,  288144661.9  ,   0.0019%
          ( 4.0 to  4.4]  ,   288145262  ,  288144660.9  ,   0.0002%
          ( 4.4 to  4.8]  ,   157043835  ,  157056047.3  ,   0.0078%
          ( 4.8 to  5.2]  ,    46607503  ,   46601762.1  ,   0.0123%
          ( 5.2 to  5.6]  ,     7508108  ,    7510327.1  ,   0.0295%
          ( 5.6 to  6.0]  ,      655295  ,     655516.1  ,   0.0337%
               >  6.0      ,       31558  ,      31686.0  ,   0.4041%
```

Rand() Example

```
#include <cstdio>//...................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//...............................................yRandom
int main(){//<=========================EXAMPLE FOR THE yRandom::Rand() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  for(int i=1;i<1000000000;++i)yRandom::Rand(I);
  printf("10^9 pseudorandom numbers generated in %.3f seconds.\nThe 10^9th"
    " number is %u.\n\n",double(clock())/CLOCKS_PER_SEC,yRandom::Rand(I));
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated in 1.778 seconds.
The 10^9th number is 2716480233.
```

RandU() Probability Density Function and Simple Example

$$f(x)=\begin{cases}\dfrac{1}{b-a}, & \text{for } a<x\le b\\ 0, & \text{otherwise}\end{cases}$$

```
#include <cstdio>//...................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//...............................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandU() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandU(I,2,6);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 8.255 seconds.
Their average is 3.999977.
```

RandN() Probability Density Function and Simple Example

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
#include <cstdio>//...................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//...............................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandN() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandN(I,4,.5);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 46.213 seconds.
Their average is 3.999998.
```

RandI() Probability Density Function and Simple Example

$$f(k)=\begin{cases}\dfrac{1}{b-a+1}, & \text{for } a\le k\le b\\ 0, & \text{otherwise}\end{cases}$$

```
#include <cstdio>//...................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//...............................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandI() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandI(I,-5,4);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 10.140 seconds.
Their average is -0.500057.
```

RandP() Probability Density Function and Simple Example

$$f(k)=\frac{\mu^k e^{-\mu}}{k!}$$

```
#include <cstdio>//...................................................printf()
#include <ctime>//.......................................clock(),CLOCKS_PER_SEC
#include "y_random.h"//...............................................yRandom
int main(){//<==================SIMPLE EXAMPLE FOR THE yRandom::RandP() FUNCTION
  unsigned I[625];/*<-*/yRandom::Initialize(I,1);//....state of Mersenne twister
  double s=0;/*<-*/for(int i=0;i<1000000000;++i)s+=yRandom::RandP(I,1);
  printf("10^9 pseudorandom numbers generated and summed in %.3f seconds.\n"
    "Their average is %f.\n\n",double(clock())/CLOCKS_PER_SEC,s/1000000000);
}//~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~21JAN2014~~~~~~
```

OUTPUT:

```
10^9 pseudorandom numbers generated and summed in 35.708 seconds.
Their average is 1.000013.
```

9. References

1. Matsumoto, M.; Nishimura, T. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation* **1998,** *8*, (1), 3–20.

2. American National Standards Institute Standard INCITS/ISO/IEC 14882-2011. *Information Technology – Programming Languages – C++* **2012**.

3. Nishimura, T; Matsumoto, M. A C-Program for MT19937, With Initialization Improved, 26 January 2002. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/CODES /mt19937ar.c. (accessed 12 December 2013).

4. Box, G. E.; Muller, M. E. A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics* **1958,** *29* (2), 610–611.

5. Abramowitz, M., Stegun, I. A., Eds. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables;* Applied Mathematics Series 55; U.S. National Bureau of Standards: Washington, DC, December 1972, section 7.1.26.

6. Knuth, D. E. *Seminumerical Algorithms, the Art of Computer Programming;* Vol. 2; Addison Wesley: Reading, MA, 1997.

1 DEFENSE TECHNICAL
(PDF) INFORMATION CTR
DTIC OCA

2 DIRECTOR
(PDF) US ARMY RESEARCH LAB
RDRL CIO LL
IMAL HRA MAIL & RECORDS MGMT

1 GOVT PRINTG OFC
(PDF) A MALHOTRA

1 RDRL WML A
(PDF) R YAGER

INTENTIONALLY LEFT BLANK.